K Medoid Algorithm

SHUBHAM PATEL

BATCH - 03

SERIAL NO. 104

Contents

Clustering

An unsupervised learning method.

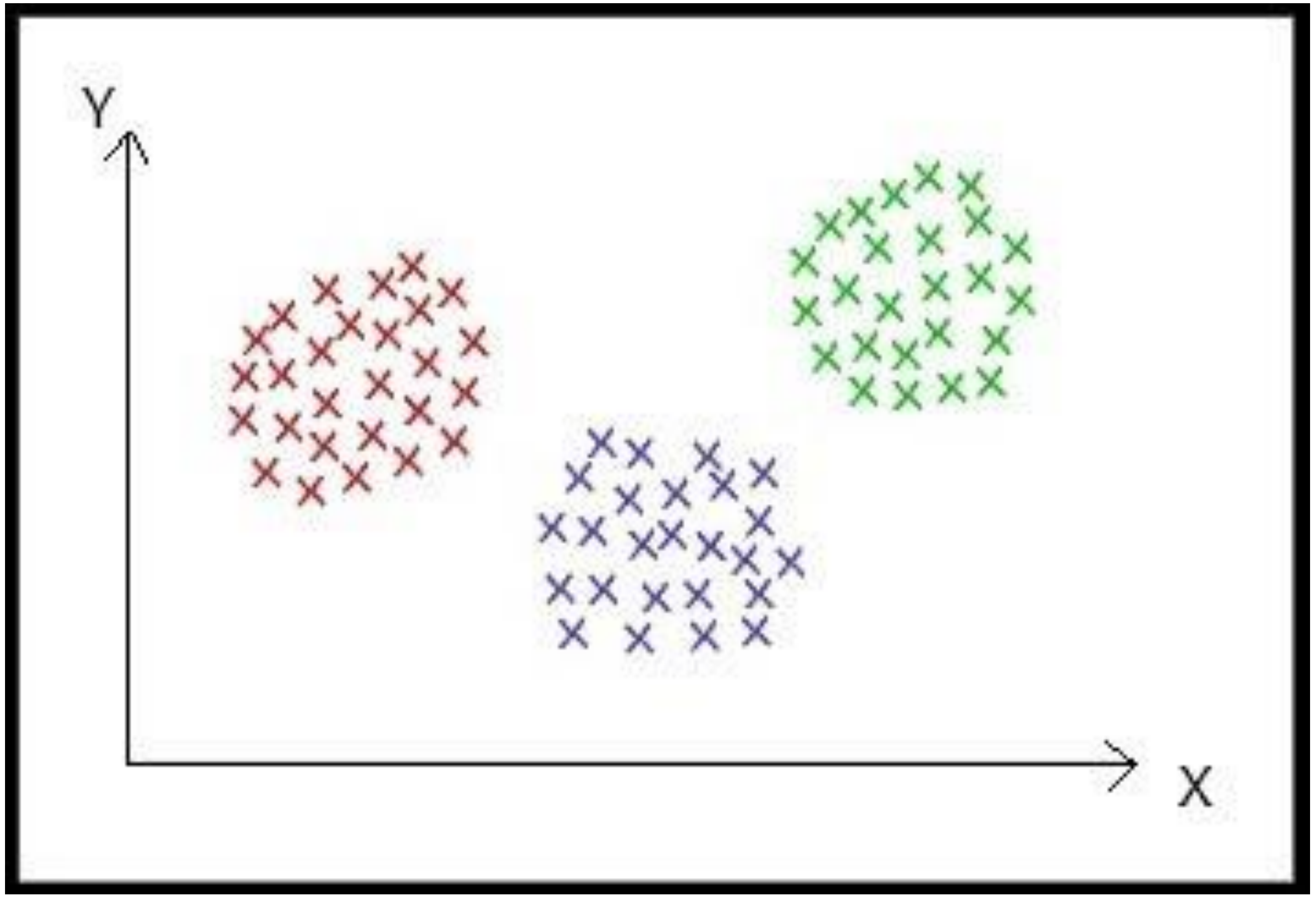
No predefined labels to cluster objects.

Types of Clustering:

- K Means clustering
- K Medoids clustering
- DBSCAN (Density bases special clustering of applications and noise)
- OPTICS (Ordering Points to identify Clustering structure)
- Hierarchical Clustering

K Medoids Clustering

K-medoids is a clustering algorithm which is median-based where we find the distance between the cluster centers and different points/objects. We assign a particular point/object to that cluster with which the distance the between point and cluster center is minimum. In K-Medoids, each cluster is represented by a medoid or the cluster median which is part of the dataset and not out of the dataset.



Algorithm/Working for K Means Clustering

Step 1: Choose the number of clusters (k) you want.
Step 2: Start with k medoids by selecting k random objects as medians.
Step 3: Compute distance from every point from medoid and cluster them accordingly.
Step 4: Adjust medoids such that they become center point of the cluster just formed.
Step 5: Again re-cluster every point based on their distance with medoids.

Continue above steps until stopping criteria is not met.

As soon as stopping criteria is met, stop and the clusters present are final clusters or the results we require.

Stopping Criteria for K Means Clustering Algorithm:

- Centroids of the newly points cluster have'nt changed
- Points in a clusters do not change their location.
- Maximum number of iterations are done.

Few important parameters of sklearn_extra.cluster.KMedoids()

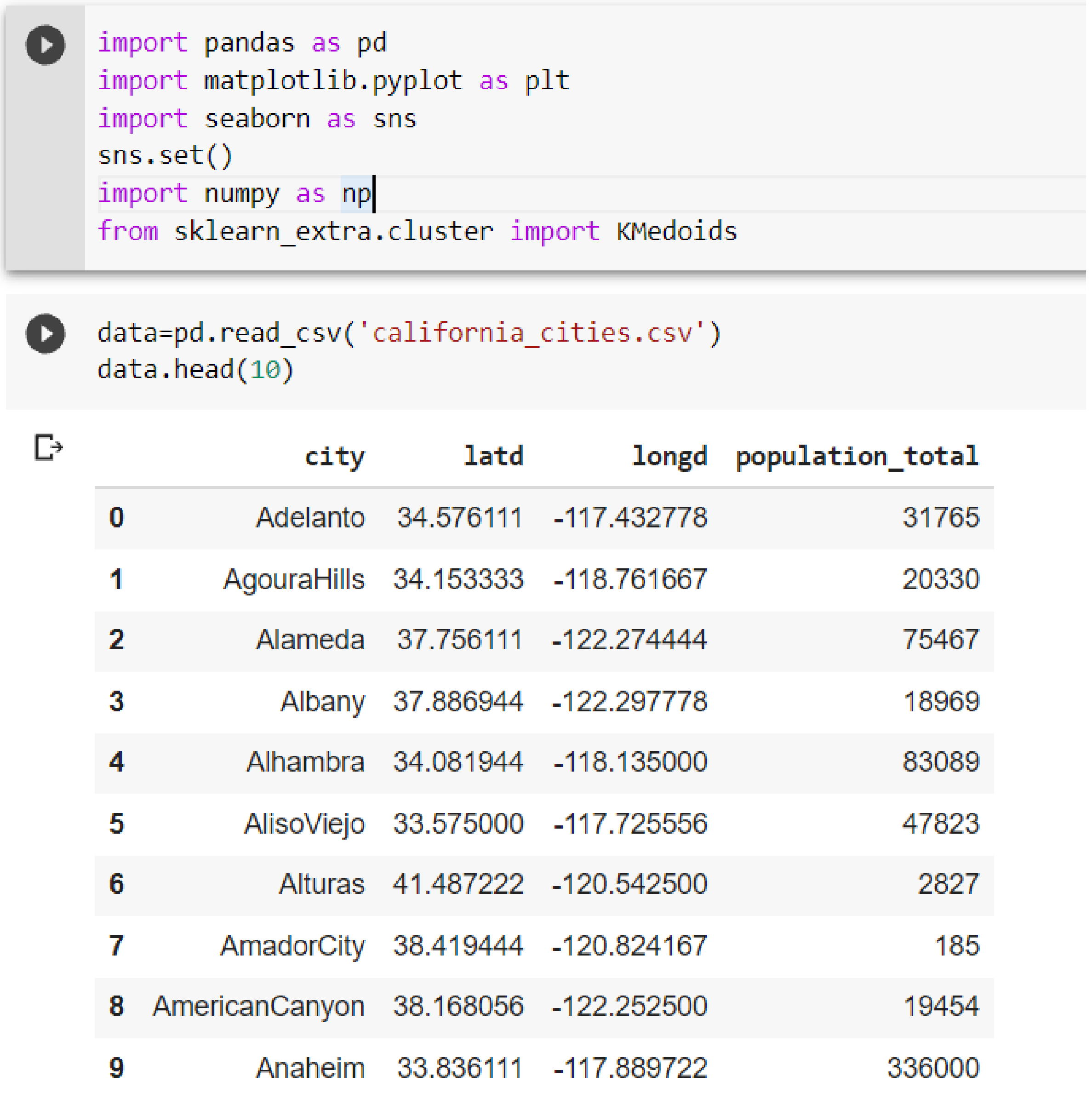
from sklearn_extra.clusters import KMedoids

- n_clusters: int, default=8
  The number of clusters to form as well as the number of medoids to generate.
- metric: string, or callable, optional, default: ‘euclidean’
  What distance metric to use
- init: {‘random’, ‘heuristic’, ‘k-medoids++’, ‘build’}, optional, default: ‘build’
  Specify medoid initialization method.
- max_iter: int, optional, default
  Specify the maximum number of iterations when fitting.
- random_state: int, RandomState instance or None, optional
  Specify random state for the random number generator

Few important attributes of sklearn_extra.cluster.KMedoids()

- cluster_centers_: array, shape = (n_clusters, n_features)
  Cluster centers, i.e. medoids (elements from the original dataset)
- medoid_indices_: array, shape = (n_clusters,)
  The indices of the medoid rows in X
- labels_: array, shape = (n_samples,)
  Labels of each point
- inertia_: float

Sum of distances of samples to their closest cluster center.



```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import numpy as np
from sklearn_extra.cluster import KMedoids
```

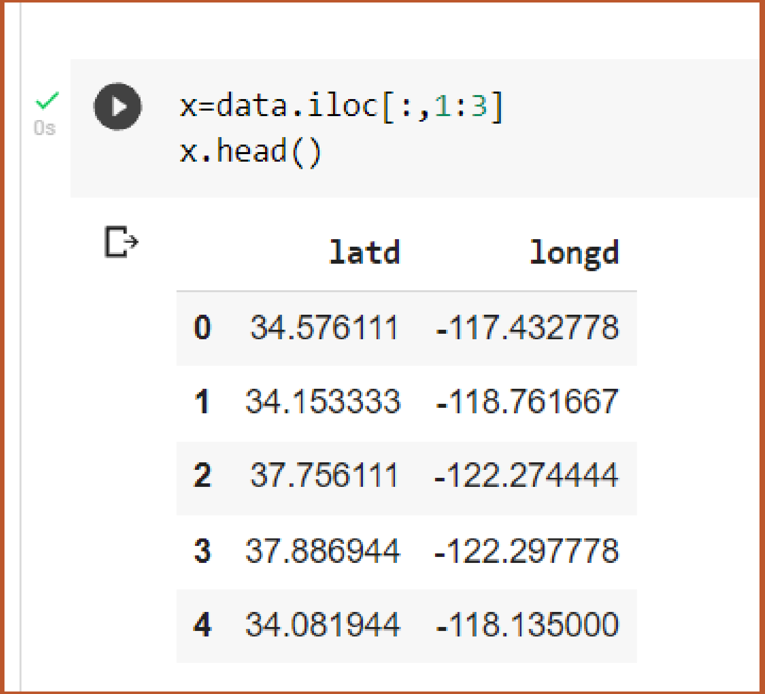
```python
data=pd.read_csv('california_cities.csv')
data.head(10)
```

```
             city       latd        longd  population_total
0        Adelanto  34.576111  -117.432778             31765
1     AgouraHills  34.153333  -118.761667             20330
2         Alameda  37.756111  -122.274444             75467
3          Albany  37.886944  -122.297778             18969
4        Alhambra  34.081944  -118.135000             83089
5      AlisoViejo  33.575000  -117.725556             47823
6         Alturas  41.487222  -120.542500              2827
7      AmadorCity  38.419444  -120.824167               185
8  AmericanCanyon  38.168056  -122.252500             19454
9         Anaheim  33.836111  -117.889722            336000
```

Implementing K-Medoid clustering

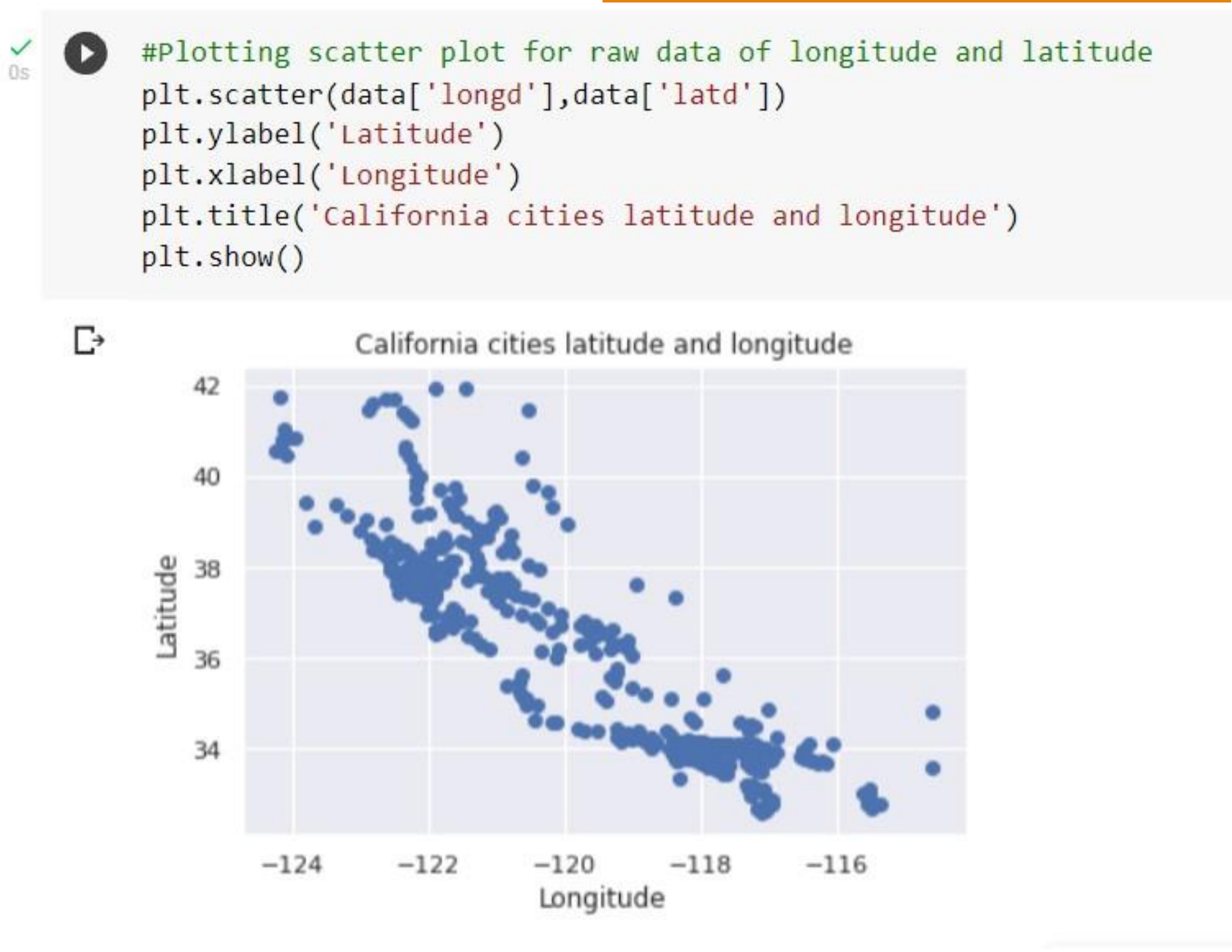
Importing required libraries and reading the dataset

Unclustered data



```
x=data.iloc[:,1:3]
x.head()
```

	latd	longd
0	34.576111	-117.432778
1	34.153333	-118.761667
2	37.756111	-122.274444
3	37.886944	-122.297778
4	34.081944	-118.135000



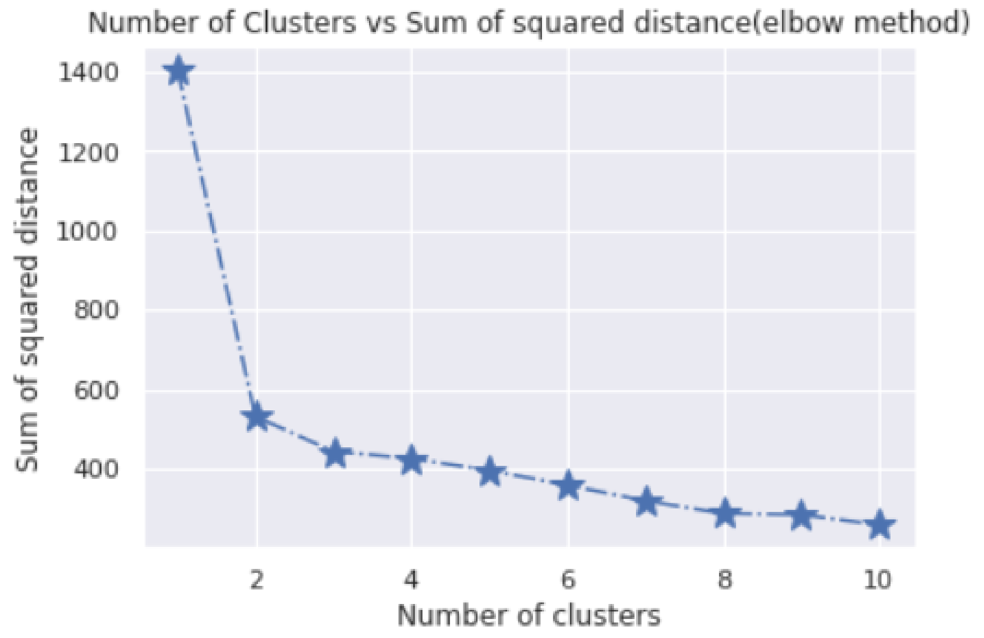
```
#Plotting scatter plot for raw data of longitude and latitude
plt.scatter(data['longd'],data['latd'])
plt.ylabel('Latitude')
plt.xlabel('Longitude')
plt.title('California cities latitude and longitude')
plt.show()
```

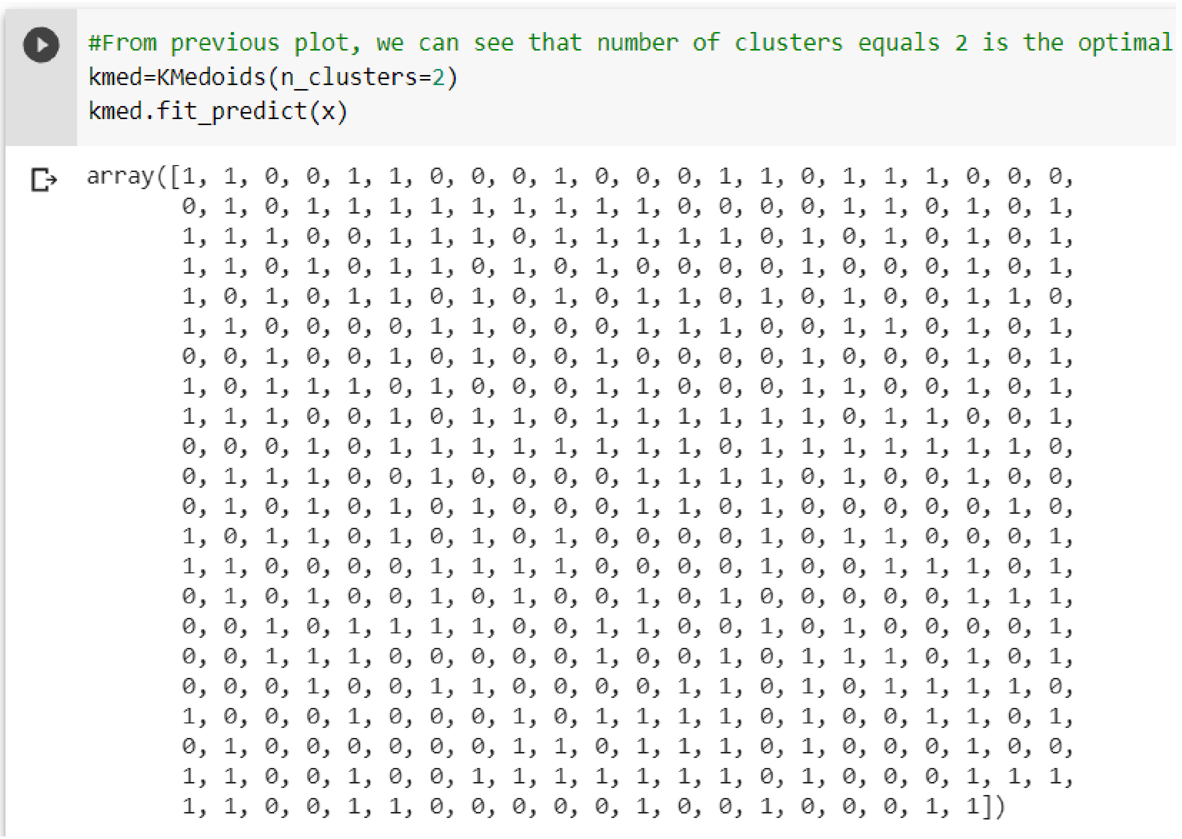
Elbow method

The elbow method helps to choose the optimum value of ‘k’ (number of clusters) by fitting the model with a range of values of ‘k’. We calculate the sum of squared distance(SSD) of each object from its cluster center, and from that, we find the number until which sharp changes in SSD are occurring. This point seems like an elbow and is the required number of k we want.

```
#Using elbow method to find optimal number of clusters
ssd=[]
for i in range(1,11):
    kmed=KMedoids(n_clusters=i,)
    kmed.fit(x)
    ssd_iter=kmed.inertia_
    ssd.append(ssd_iter)
```

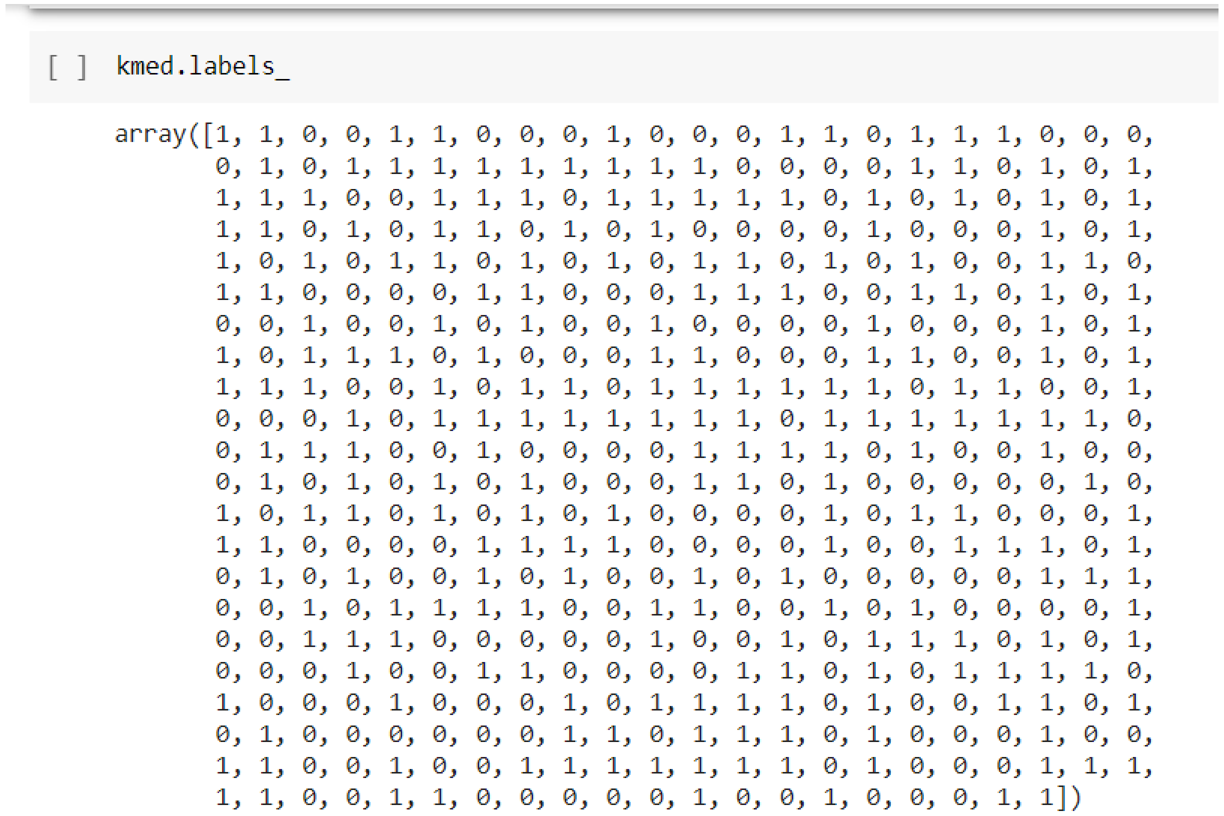
```
number_clusters=range(1,11)
plt.plot(number_clusters,ssd,marker='*',linestyle='-.',markersize=15)
plt.title('Number of Clusters vs Sum of squared distance(elbow method)')
plt.xlabel('Number of clusters')
plt.ylabel('Sum of squared distance')
plt.show()
```





```
#From previous plot, we can see that number of clusters equals 2 is the optimal
kmed=KMedoids(n_clusters=2)
kmed.fit_predict(x)
```

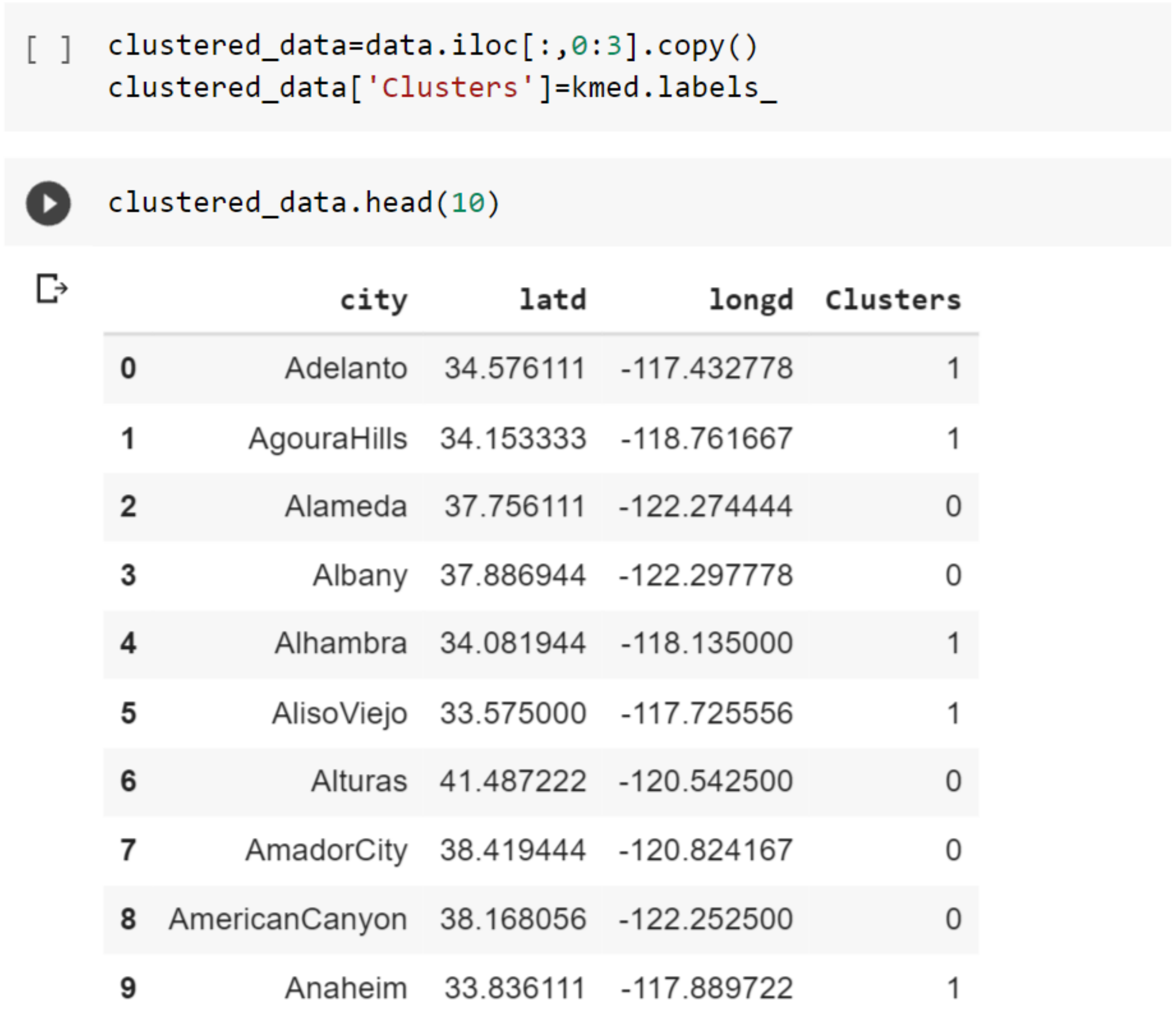
```
array([1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1])
```



```
[ ]  kmed.labels_
```

```
array([1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1])
```

Using KMediods() to create model



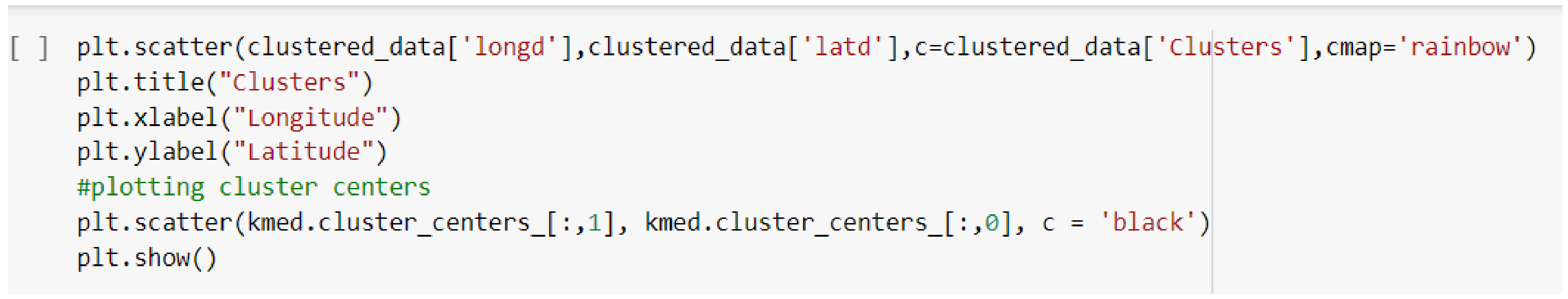
```
[ ] clustered_data=data.iloc[:,0:3].copy()
    clustered_data['Clusters']=kmed.labels_
```

```
clustered_data.head(10)
```

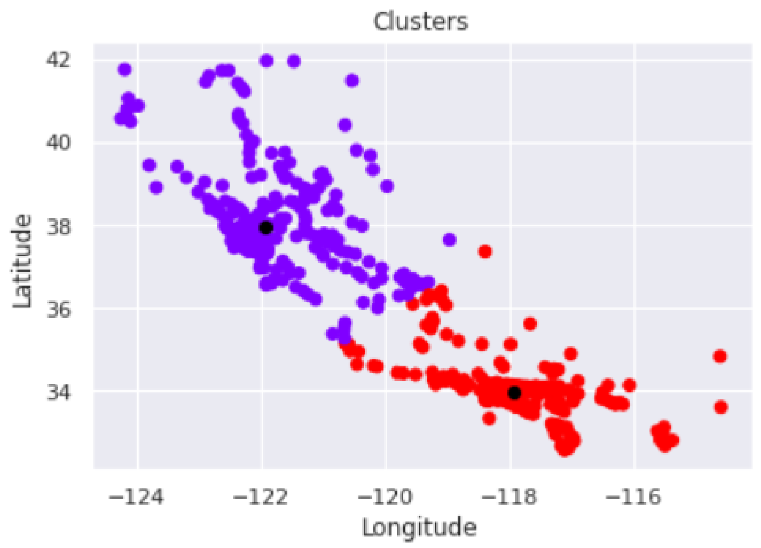
	city	latd	longd	Clusters
0	Adelanto	34.576111	-117.432778	1
1	AgouraHills	34.153333	-118.761667	1
2	Alameda	37.756111	-122.274444	0
3	Albany	37.886944	-122.297778	0
4	Alhambra	34.081944	-118.135000	1
5	AlisoViejo	33.575000	-117.725556	1
6	Alturas	41.487222	-120.542500	0
7	AmadorCity	38.419444	-120.824167	0
8	AmericanCanyon	38.168056	-122.252500	0
9	Anaheim	33.836111	-117.889722	1

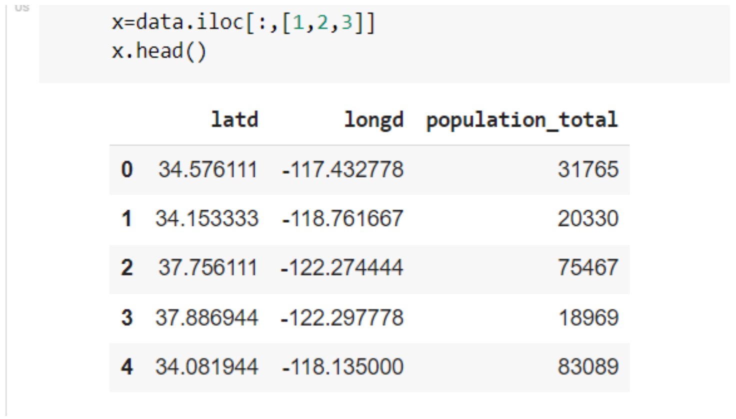
Clustered Data after assigning to Clusters

Plot scatter plot for clustered data



```
[ ]  plt.scatter(clustered_data['longd'],clustered_data['latd'],c=clustered_data['Clusters'],cmap='rainbow')
     plt.title("Clusters")
     plt.xlabel("Longitude")
     plt.ylabel("Latitude")
     #plotting cluster centers
     plt.scatter(kmed.cluster_centers_[:,1], kmed.cluster_centers_[:,0], c = 'black')
     plt.show()
```

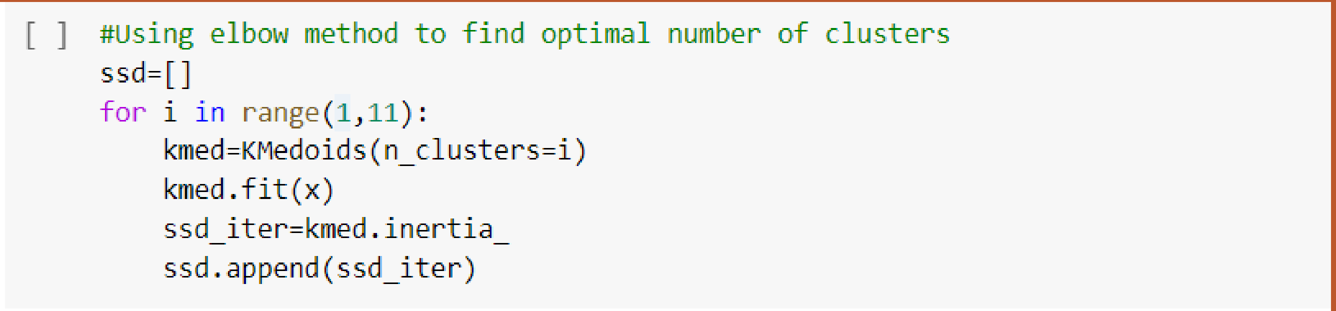




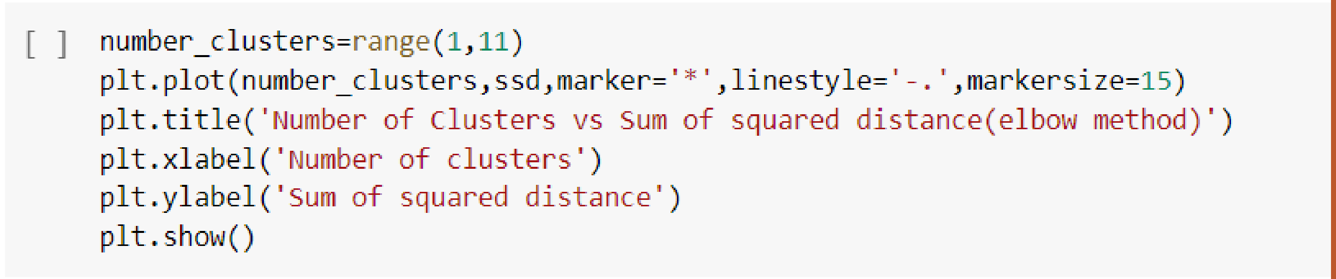
```
x=data.iloc[:,[1,2,3]]
x.head()
```

	latd	longd	population_total
0	34.576111	-117.432778	31765
1	34.153333	-118.761667	20330
2	37.756111	-122.274444	75467
3	37.886944	-122.297778	18969
4	34.081944	-118.135000	83089

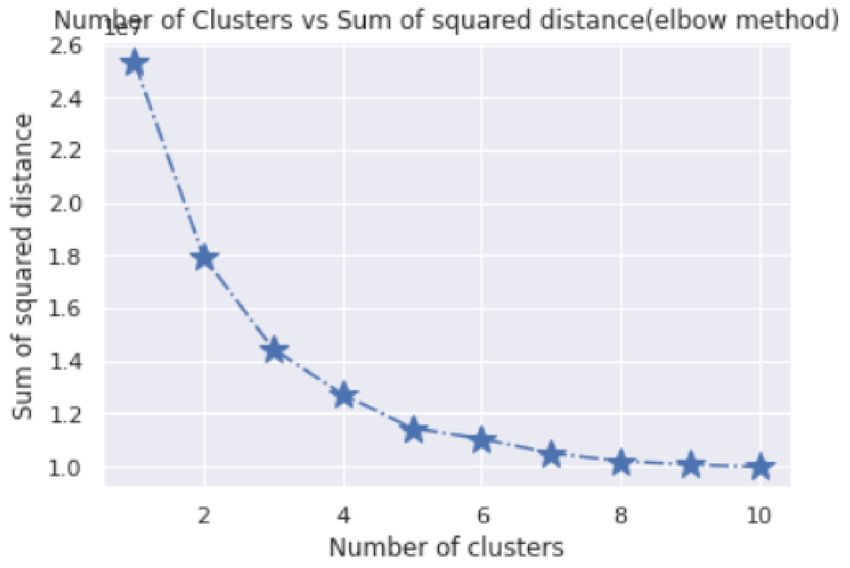
Cluster data on basis of 3 or more variable

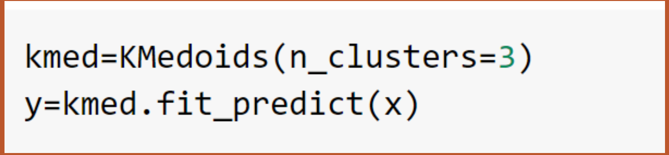


```
[ ] #Using elbow method to find optimal number of clusters
    ssd=[]
    for i in range(1,11):
        kmed=KMedoids(n_clusters=i)
        kmed.fit(x)
        ssd_iter=kmed.inertia_
        ssd.append(ssd_iter)
```

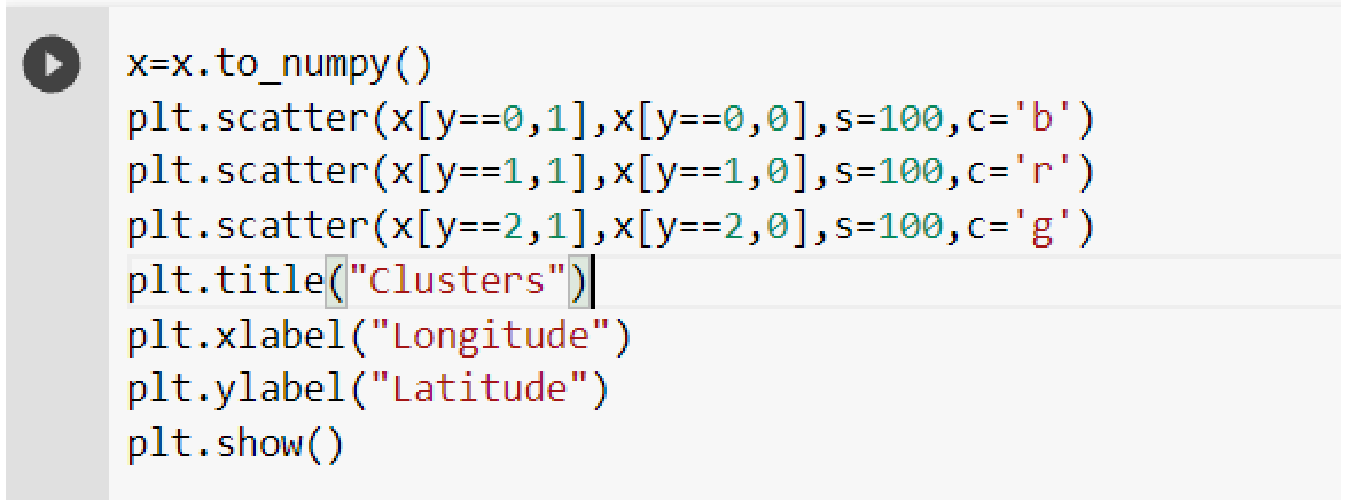


```
[ ] number_clusters=range(1,11)
    plt.plot(number_clusters,ssd,marker='*',linestyle='-.',markersize=15)
    plt.title('Number of Clusters vs Sum of squared distance(elbow method)')
    plt.xlabel('Number of clusters')
    plt.ylabel('Sum of squared distance')
    plt.show()
```

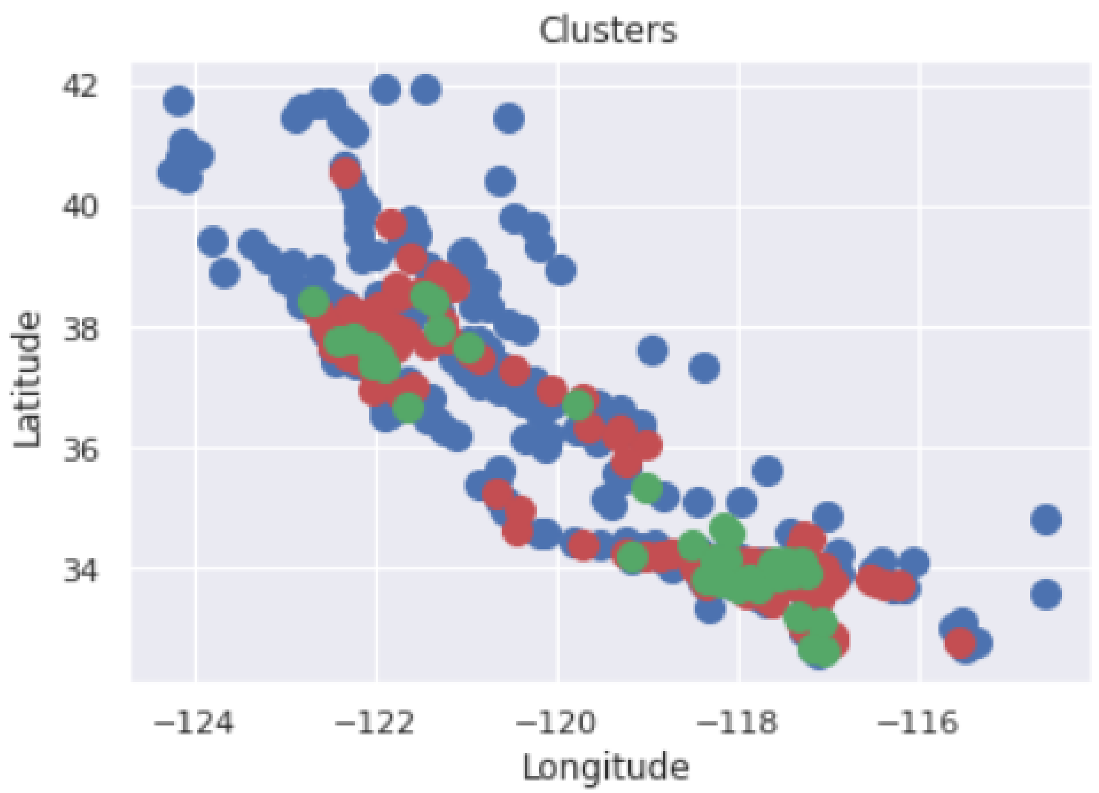




```
kmed=KMedoids(n_clusters=3)
y=kmed.fit_predict(x)
```



```
x=x.to_numpy()
plt.scatter(x[y==0,1],x[y==0,0],s=100,c='b')
plt.scatter(x[y==1,1],x[y==1,0],s=100,c='r')
plt.scatter(x[y==2,1],x[y==2,0],s=100,c='g')
plt.title("Clusters")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```



Plot scatter plot for clustered data

Applications of K medoids clustering

Firstly, the courier company example for deciding where to open delivery center :)

It is used in the clustering of documents to identify the compatible documents in the same place.

It is helpful in the business sector for recognizing the portions of purchases made by customers, also to cluster movements on apps and website.

Clustering forms a backbone of search engines. When a search is performed, the search results need to be grouped, and the search engines very often use clustering to do this.

There are many more applications. You can always refer internet to know more

Advantages and Disadvantages of K Mediods clustering

ADVANTAGES

- It is simple to understand and easy to implement.
- K-Medoid Algorithm is fast and converges in a fixed number of steps.
- K Mediod algorithm is less sensitive to outliers than other partitioning algorithms.

DISADVANTAGES

- Sometimes, it is quite tough to forecast the number of clusters, or the value of k.
- It may obtain different results for different runs on the same dataset because the first k medoids are chosen randomly.
- It is not directly applicable to categorical data

Thankyou